### **JS EVENTS**

### **1. Event Types**

Here are some common types of events:

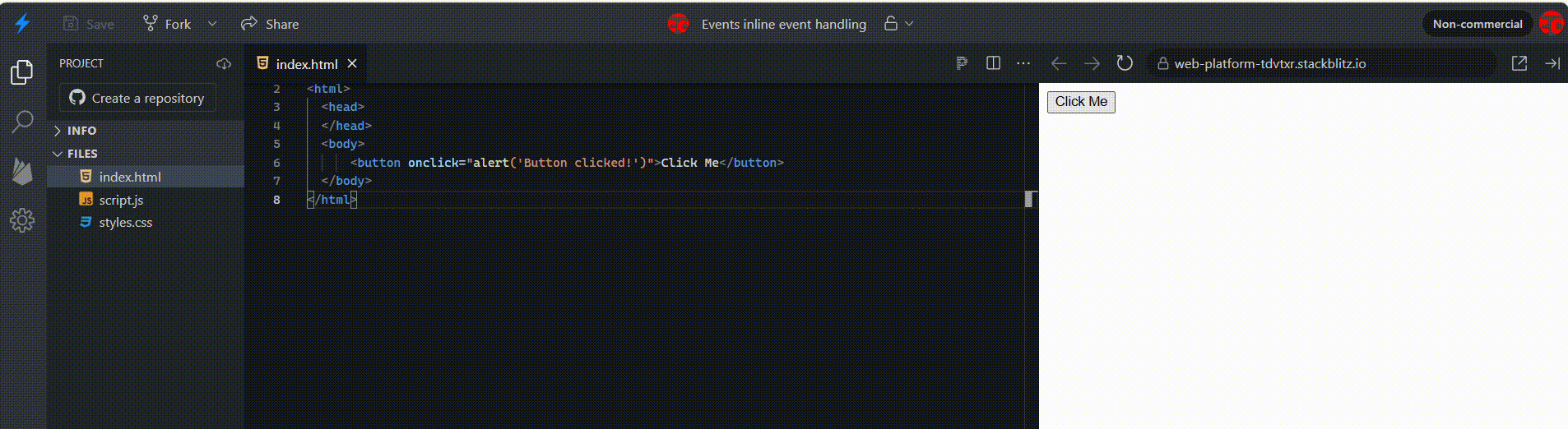
* **Mouse Events**: click, dblclick, mousedown, mouseup, mouseover, mouseout, mousemove
* **Keyboard Events**: keydown, keyup, keypress
* **Form Events**: submit, change, input, focus, blur
* **Window Events**: load, resize, scroll, beforeunload
* **Touch Events**: touchstart, touchmove, touchend

### **2. Event Handlers**

Event handlers are functions that are executed in response to specific events. You can add event handlers in several ways:

#### **Inline Event Handlers**

You can directly add event handlers in your HTML markup using attributes.



**Explanation of how the above code works:**

<button onclick="alert('Button clicked!')">Click Me</button>

**<button> Element**: This creates a clickable button on a webpage.

**onclick Attribute**: This attribute specifies a JavaScript action that should be executed when the button is clicked.

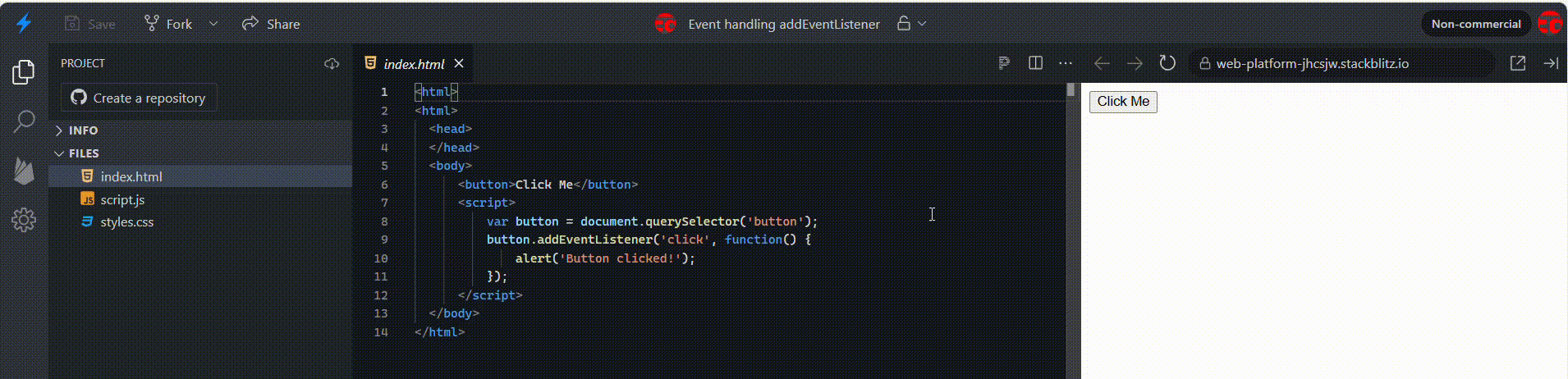
**alert('Button clicked!')**: This is the JavaScript code that runs when the button is clicked. The alert function displays a popup dialog box with the message "Button clicked!".

**Open this url to check the code and try yourself:**

[**https://stackblitz.com/edit/web-platform-tdvtxr?file=index.html**](https://stackblitz.com/edit/web-platform-tdvtxr?file=index.html)

#### **Using JavaScript addEventListener()**

The addEventListener() method allows you to attach multiple event handlers to an element.



**Explanation of how the above code works:**

**var button = document.querySelector('button');**

**This line selects the first <button> element in the document and assigns it to the variable button. The document.querySelector() method uses a CSS selector to find the element.**

**button.addEventListener('click', function() {**

**This line adds an event listener to the button element. The addEventListener() method takes two arguments:**

**1.The type of event to listen for ('click' in this case).**

**2.A callback function that will be executed when the event occurs. Here, an anonymous function (defined using the function keyword) is provided.**

**alert('Button clicked!');**

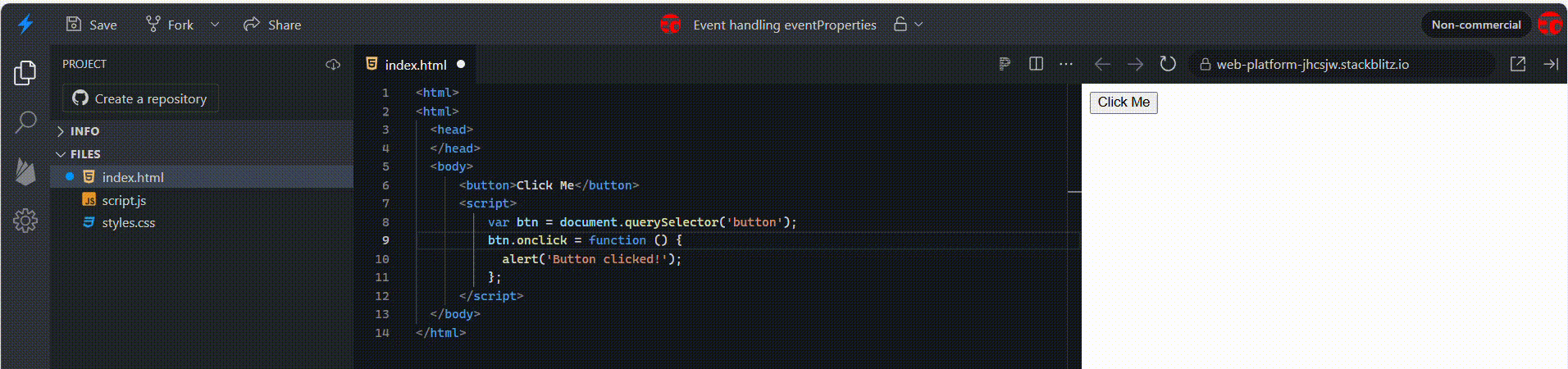
**Inside the callback function, this line triggers an alert dialog that displays the message "Button clicked!" when the button is clicked.**

**Open this url to check the code and try yourself:**

[**https://stackblitz.com/edit/web-platform-ipfhui?file=index.html**](https://stackblitz.com/edit/web-platform-ipfhui?file=index.html)

#### **Using Event Properties**

You can also set event handlers directly as properties of DOM elements.



**Explanation of how the above code works:**

**var btn = document.querySelector('button');**

**This line selects the first <button> element in the document and assigns it to the variable btn. The document.querySelector() method is used to find elements using a CSS selector.**

**btn.onclick = function ()**

**This line assigns a function to the onclick property of the btn variable. The onclick property serves as an event handler that defines what should happen when the button is clicked. Here, an anonymous function is used.**

**alert('Button clicked!');**

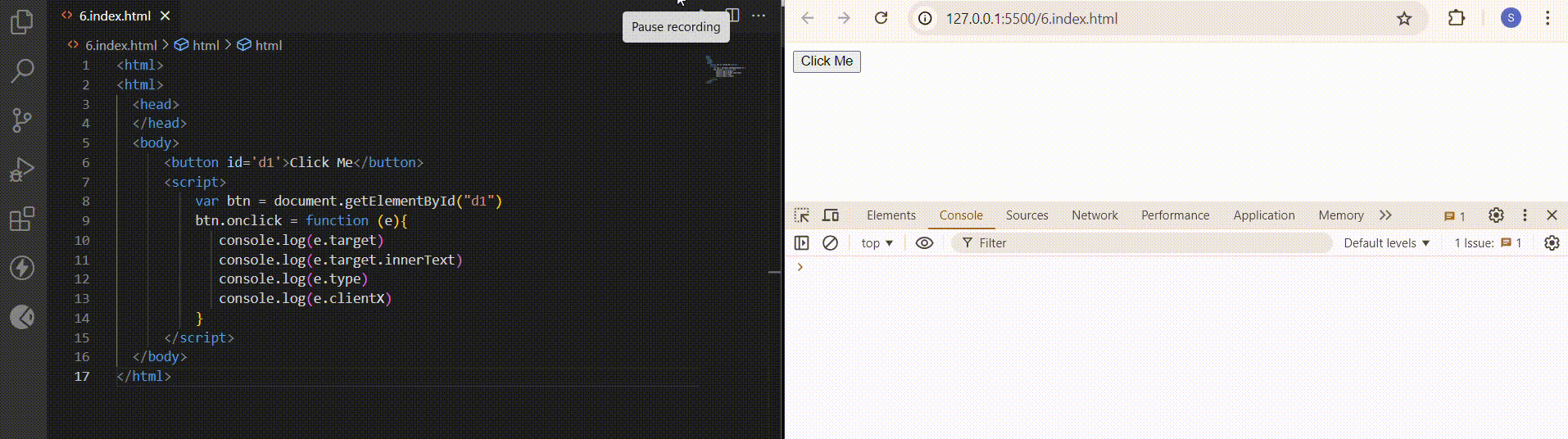
**Inside the function, this line triggers an alert dialog that displays the message "Button clicked!" whenever the button is clicked.**

**Open this url to check the code and try yourself:**

[**https://stackblitz.com/edit/web-platform-jhcsjw?file=index.html**](https://stackblitz.com/edit/web-platform-jhcsjw?file=index.html)

### **3. Event Object**

When an event occurs, an event object is automatically passed to the event handler. This object contains information about the event.



**Explanation of how the above code works:**

**var btn = document.getElementById("d1");**

**This line selects the HTML element with the ID "d1" and assigns it to the variable btn. The document.getElementById() method retrieves the element based on its unique ID.**

**btn.onclick = function (e)**

**This line assigns an anonymous function to the onclick property of the btn variable. The onclick property serves as an event handler that will execute when the element is clicked. The parameter e is the event object, which contains information about the event.**

**console.log(e.target);**

**This line logs the element that triggered the event (in this case, the button itself) to the console. The e.target property references the specific element that was clicked.**

**console.log(e.target.innerText);**

**This line logs the text content of the clicked element to the console. The innerText property retrieves the visible text inside the element.**

**console.log(e.type);**

**This line logs the type of event that occurred (in this case, 'click') to the console. The e.type property indicates what kind of event was triggered.**

**console.log(e.clientX);**

**This line logs the horizontal coordinate (in pixels) of the mouse pointer relative to the viewport when the event occurred. The clientX property gives you the X position of the mouse cursor at the time of the click.**

**Open this url to check the code and try yourself:**

[**https://stackblitz.com/edit/web-platform-aizkd2?file=index.html**](https://stackblitz.com/edit/web-platform-aizkd2?file=index.html)

### **4. Event Propagation**

Events in the DOM propagate in two phases:

* **Capturing Phase**: The event starts from the root and propagates down to the target element.
* **Bubbling Phase**: The event bubbles up from the target element to the root.

You can control event propagation using:

#### **event.stopPropagation()**

Prevents the event from propagating further.

document.querySelector('button').addEventListener('click', (event) => {

event.stopPropagation();

console.log('Button clicked!');

});

#### **event.stopImmediatePropagation()**

Stops the event from propagating and prevents any other event handlers from being executed on the same element.

document.querySelector('button').addEventListener('click', (event) => {

event.stopImmediatePropagation();

console.log('Button clicked!');

});

#### **event.preventDefault()**

Prevents the default action associated with the event (e.g., form submission).

document.querySelector('form').addEventListener('submit', (event) => {

event.preventDefault();

console.log('Form submission prevented!');

});

### **5. Event Delegation**

Event delegation involves attaching a single event handler to a parent element to handle events for its child elements. This is efficient and helps manage events for dynamically added elements.

document.querySelector('#parent').addEventListener('click', (event) => {

if (event.target && event.target.matches('.child')) {

alert('Child element clicked!');

}

});

### **6. Creating Custom Events**

You can create and dispatch your own events using CustomEvent.

#### **Creating and Dispatching Custom Events**

const event = new CustomEvent('myCustomEvent', { detail: { key: 'value' } });

document.dispatchEvent(event);

document.addEventListener('myCustomEvent', (e) => {

console.log(e.detail); // Output: { key: 'value' }

});

### **7. Event Listener Options**

When adding event listeners, you can specify options such as capture and once.

* **capture**: If true, the event is captured during the capturing phase.
* **once**: If true, the event handler will be invoked at most once.

button.addEventListener('click', () => {

console.log('Button clicked!');

}, { once: true });

### **8. Removing Event Listeners**

You can remove an event listener using removeEventListener(). The handler function must be the same as the one passed to addEventListener().

const handleClick = () => {

alert('Button clicked!');

};

button.addEventListener('click', handleClick);

button.removeEventListener('click', handleClick);

### **9. Keyboard Events**

Keyboard events provide information about keyboard interactions.

document.addEventListener('keydown', (event) => {

console.log(`Key pressed: ${event.key}`);

});

### **10. Mouse Events**

Mouse events provide information about mouse interactions.

document.addEventListener('mousemove', (event) => {

console.log(`Mouse position: (${event.clientX}, ${event.clientY})`);

});

### **Summary**

Events in JavaScript are essential for creating interactive web applications. They allow you to respond to user interactions and other activities in the browser. By understanding and leveraging various event types, handlers, and propagation techniques, you can build more dynamic and engaging user experiences.